

Installation de MINGW

MinGW Kesako ?

Rappel !

Lorsque vous codez dans un langage non interprété (comme le C, pas comme le python) vous transformez du code texte en code machine (binaire), pour cela vous faites appel à un compilateur, un programme qui est chargé d'effectuer la dure tâche de transformer ce texte dans un code binaire qui a du sens pour votre processeur. Ce code a directement un sens d'un point de vue électrique, ce sont des parties d'un circuit qui s'active ou se désactive pour aboutir à un résultat (voir vos cours d'électronique et physiques sur les transistors, etc., ainsi que vos cours de Culture numérique).

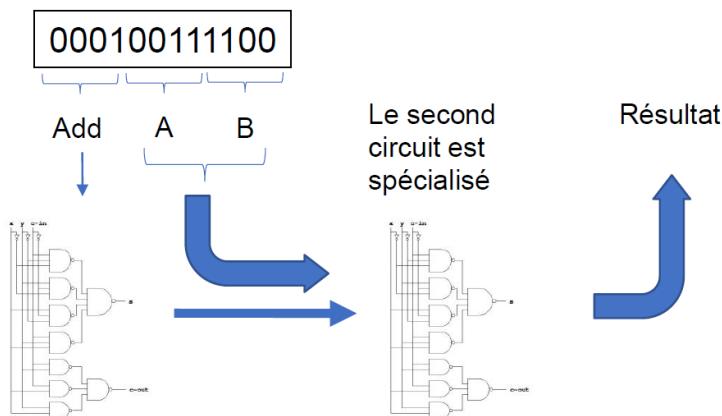


Figure 1: Exemple de lecture d'un code binaire : la première partie décrit une opération (ici l'addition) la seconde décrit les informations à traiter par l'opération (ici deux chiffres en binaire). L'opération est lue par un circuit qui choisit le circuit qui l'effectuera puis elle est ensuite envoyée pour être exécuté par un circuit spécialisé.

Forcément, chaque processeur étant un ensemble de circuits différent, le code binaire pour leur communiquer une même opération est parfois différent. De fait, le travail du compilateur est de comprendre comment prendre en compte votre matériel pour adapter le résultat en fonction.

De même, le système d'exploitation (Windows, Linux, etc.) qui est un programme général (écrit en C aussi 😊) qui cherche principalement à gérer les autres programmes et l'accès au matériel (disque dur, mémoire, etc.) impose qu'on communique avec lui pour avoir certaines autorisations (écrire en mémoire, etc.), ce qui là encore, passe par du code différent en fonction du système d'exploitation. C'est également le travail du compilateur de s'adapter au système d'exploitation sur lequel il compile.

Il existe donc une multitude de résultats différents sortant du même compilateur, mais il existe aussi une multitude de compilateurs différents, certains sont spécialisés dans un domaine ou dans un autre, d'autres plus généralistes.

Il est important que vous connaissiez ces quelques compilateurs :

- GCC
- CLANG
- MSVC

Clang et GCC sont issus du monde du logiciel libre, et donc plutôt proche à l'origine du système d'exploitation Linux. Mais aujourd'hui on peut les exécuter sous de nombreuses architectures, dont Windows. MSVC est le compilateur Windows natif, il ne fonctionne que sous Windows. C'est lui qui est livré en standard avec **Visual Studio**, l'éditeur de code de Microsoft.

Pourquoi MinGW ?

GCC est le compilateur qui nous intéresse depuis le début, c'est en effet un compilateur issu du monde libre, donc disponible partout, très généraliste, donc adapté à de très nombreuses architectures, allant des objets embarqués aux serveurs en passant par la robotique, etc., et disponible autant en C qu'en C++. On le préfère à vous habituer à MSVC, plus spécifique Windows.

Malheureusement, GCC a été codé à l'origine principalement pour Linux, et pour fonctionner sous Windows il faut donc qu'il ait accès à certains outils existants sous Linux, mais qui n'existaient pas sous Windows. Il fallait également écrire une version Windows spécifique, capable de parler avec le système Windows, différente de la version Linux. C'est à ça que sert MinGW, c'est le « **MINimalist Gcc for W**indows ».

Au départ, MinGW n'a été conçu que pour fournir GCC, mais au fur et à mesure il a inclus un ensemble d'outils venant du monde de Linux qui ont été réécrits pour fonctionner sous Windows (on peut alors utiliser des choses comme nano, vi, vim, etc.).



Le saviez-vous ?

Microsoft s'est peu à peu rendu compte que les outils Linux étaient plus conviviaux et plus utilisés. De fait, depuis Windows 10, le « Powershell » est un nouveau terminal Windows qui intègre peu à peu pas mal de commande Linux, tel que « ls » par exemple.

Pour installer GCC sous Windows il est donc nécessaire d'installer MinGW qui inclut GCC version Windows comme certains outils pratiques que vous avez l'habitude d'utiliser sous Linux (ou Mac, car les Mac sont basés sur un noyau Unix, parent de Linux).

Différence avec WSL ?

La question que vous pouvez vous poser, c'est « *Si je peux coder sous Windows depuis le début, pourquoi ai-je installé WSL, c'est quoi la différence ?* ».

WSL est un mini système Linux à part entière. Quand vous êtes sous WSL vous n'êtes pas sous Windows, mais sous un (presque) véritable Linux. Un peu comme si vous aviez deux systèmes à part entière en parallèle.



De fait, avec tout ce que vous savez maintenant, il est évident qu'un programme compilé sous WSL ne sera pas exécutable directement depuis votre Windows : si vous le trouvez dans Windows et que vous faites un double clic dessus, ça ne fonctionnera pas.

Donc il y a bien une différence majeure entre les deux approches. GCC MinGW vous permet de créer un programme .exe windows parfaitement natif, qui s'exécutera comme tout programme Windows. Alors que GCC sous WSL crée un programme Linux, qui ne fonctionnera que sous des systèmes Linux (vrai Linux ou WSL).

En pratique !

Téléchargement et installation :

L'installation de MinGW est assez simple, mais il y a de nombreuses approches. La plus efficace pour nous est de partir sur l'installation d'un package « tout en un » nommé **MSYS2**.

MSYS2 étend encore l'idée de MinGW en ajoutant de nombreux packages Linux nativement sous Windows, disponible en une ligne de commande dans le terminal MSYS2 qui se comportera comme le fameux « sudo apt install » que vous connaissez bien sous Linux.

- Commençons par télécharger MSYS 2 à cette adresse :
<https://www.msys2.org/>
- Lancez l'installateur de MSYS2 (msys2-x86_64-20240113.exe) et choisissez un dossier où l'installer qui est à un nom court, simple, proche du disque C. Nous recommandons « **C:\msys2** »
- Quand vous avez fini, vous pouvez cocher « run MSYS2 ».
- Taper ensuite **pacman -Suy** dans le terminal pour mettre à jour l'ensemble du système. (pacman est ici l'équivalent de apt sous Debian, le formalisme est un peu différent, mais le principe est identique)

Installation des packages :

MSYS2 tout seul n'est qu'un terminal avec très peu d'outils. Installons quelques outils élémentaires pour pouvoir coder dans de bonnes conditions. Dont GCC bien entendu.

Les outils sont proposés en deux formats/options :

- UCRT-x86_64
- x86-64 tout seul

C'est dû au fait que le système Windows a évolué drastiquement entre Windows 7 et Windows 10. Les bibliothèques permettant l'exécution des programmes ont évolué de l'ancien système nommé MSVCRT (Microsoft Visual C++ Redistributable) à UCRT (Universal C Runtime). Pour plus de détails, consultez : <https://www.msys2.org/docs/environments/>

Comme on est tous sous des versions de Windows supérieur à Windows 10, nous pouvons installer les versions UCRT :

- Vous pouvez taper dans le terminal ouvert par MSYS2 :
« **pacman -S mingw-w64-ucrt-x86_64-toolchain** »

Ce package équivaut à installer les outils suivants de développement :

- GCC
- Git
- GDB (pour debugger)
- Make (pour le makefile)
- PKGCONF (pas utile pour nous ici)

S'il s'avère que les outils ci-dessus ne sont pas installés avec la commande précédente, veuillez installer manuellement les commandes de la façon suivante :

« **pacman -S gcc** »
« **pacman -S make** »
...

- Vous pouvez installer d'autres package si vous avez envie, si vous désirez un package de Linux et souhaitez vérifier qu'il existe sous MSYS2, tapez simplement dans google « le nom de mon package UCRT64 msys2 », vous devriez arriver en premier résultat sur la page vous indiquant le nom du package et divers détails, ce qui vous permettra d'utiliser ce nom pour la commande pacman.

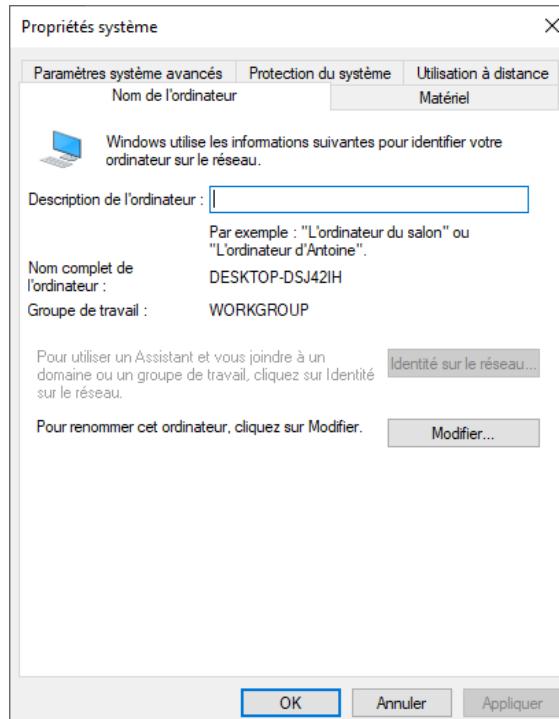
Variables d'environnement :

Une fois tout ceci fait, ce n'est pas tout à fait fini. En effet, les exécutables que vous installez via les commandes de Msys2 sont installés dans le dossier de Msys2 présent, si vous avez choisi **C:\msys2**, à l'emplacement suivant : **C:\msys2\ucrt64\bin**

Il faut informer Windows que nous allons travailler avec ce dossier pour qu'il aille y chercher les exécutables quand on tape les noms dans le terminal.

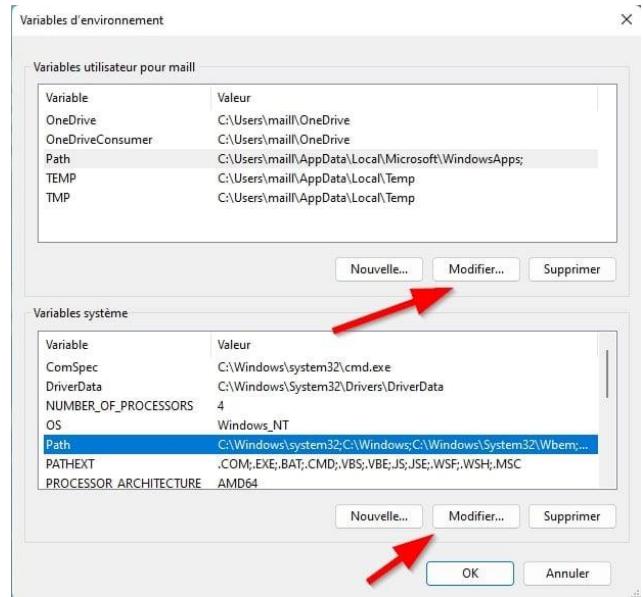
Pour ce faire, il faut modifier les variables d'environnements système. Vous pouvez taper « environnement » dans la recherche Windows et il vous le propose immédiatement.

Sinon taper **sysdm.cpl** dans la barre d'exécution/recherche du menu démarrer. Vous arriverez dans les deux cas sur cette fenêtre :

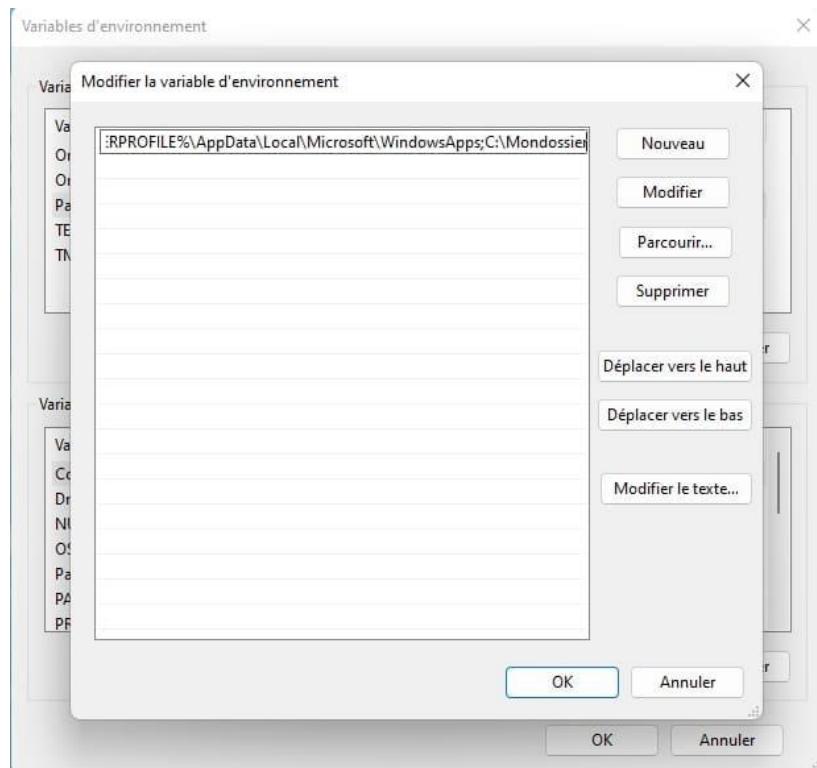


Faites attention à partir de cette étape, cette manipulation est dangereuse, suivez scrupuleusement les consignes !

Et vous pourrez alors sélectionner « paramètre système avancé » en haut, puis « variable d'environnement ». À partir de là sélectionner la variable path (en haut pour vous uniquement, ou en bas pour tous les utilisateurs du système), puis cliquez sur modifier.



Vous arrivez sur la fenêtre suivante :



Ajoutez simplement une nouvelle ligne en appuyant sur « nouveau » et dans cette ligne vous pouvez copier-coller le nom du dossier précédent :

C:\msys2\ucrt64\bin

(Si vous avez choisi msys2 comme nom de dossier).

À partir de là vous pouvez redémarrer windows, puis ouvrir un powershell (taper simplement « powershell dans la recherche windows ou même dans la barre d'adresse de n'importe quel dossier, vous pouvez ouvrir une invite de commande avec un clic droit également sous Windows 11), puis vérifier que gcc fonctionne en tapant gcc.exe --version.

Et voilà, vous êtes fin prêt pour développer sous Windows !